PIXIE-4 Online Help

Version 1.61, May 2008

XIA LLC

31057 Genstar Road
Hayward, CA 94544 USA

Phone: (510) 401-5760; Fax: (510) 401-5761
http://www.xia.com

Disclaimer

Information furnished by XIA is believed to be accurate and reliable. However, XIA assumes no responsibility for its use, or for any infringement of patents, or other rights of third parties, which may result from its use. No license is granted by implication or otherwise under the patent rights of XIA. XIA reserves the right to change the DGF product, its documentation, and the supporting software without prior notice.

• Getting Help for the Pixie-4

There are several ways to get help for the Pixie-4. You can use IGOR's built-in help browser to access the Pixie-4 specific help file by selecting Help -> Help Topics from the top menu bar. Choose "Pixie4-Help" in the popup menu on the left, and select the appropriate help topic from the list on the right.

Each Pixie-4 Run Control Panel also has a "Help" button, which directly displays the help topic for that panel. In the help topics, click on blue underlined links to jump to cross references.

• Getting Started

Preparations

1. If you have a remote controller (PCI bridge),
   a) first install the driver software for the controller itself on the host PC. For NI controllers, typically this means installing the default software from the "Device Driver Reference CD"
   b) power down the host computer, install the PCI bridge in both the host computer and chassis, and connect them with the bridge cable.
   c) power up the system again (chassis first).

2. Install Wavemetrics IGOR Pro on the host computer (or embedded PC).

3. Install the Pixie-4 software from XIA (see file readme.txt on CD-ROM).

4. If not already installed, install the Pixie-4 modules in the PXI/CompactPCI crate with the power switched off. Connect detector signals to the Pixie-4 modules using the cables supplied by XIA. Then switch on power of the chassis. For remote controllers, you have to reboot the PC after power cycling the chassis.

6. After the host PC booted with Pixie-4 modules present, it will report new hardware found. When Windows is looking for driver files, direct it to the "drivers" folder in the XIA software distribution. A properly installed Pixie-4 module should be listed in Window's Device Manager under "Other devices" as a "Custom (OEM) PCI 9054 board".

7. Find *Pixie4.pxp* in the installed folder and double-click it to open the Pixie-4 Viewer.

Initial Startup

When the Pixie Viewer has been loaded, the *Pixie Start Up Panel* should be prominently displayed in the middle of the desktop. It will prompt you to do the following:

Select PXI chassis type
First select the PXI chassis you are using. Currently supported are standard 4-slot or 8-slot chassis and the National Instrument 14-slot or 18-slot chassis with 3 PCI bus segments. (The 14-slot chassis has the same PCI bus configuration as the 18-slot chassis, only fewer slots)

Specify the Pixie-4 modules:
Next select the number of Pixie-4 modules in the system. Then specify the PXI slot number in which each module resides.
Note: If using the 18-slot chassis, the Pixie-4 module designated as module 0 must reside in the first bus segment (any of slots 1-6).

Finally, click [**Start Up System**]. In the IGOR history window, a message will show if the modules have been initialized successfully. You will now see the main *Pixie4 Run Control* Panel from which all work

© XIA 2008. All rights reserved.

is conducted. The tabs in the *Pixie4 Run Control* Panel are arranged in logical order from left to right. For most of the actions the Pixie Viewer interacts with one Pixie-4 module at a time. The number of that module is displayed at the top right corner of the *Pixie4 Run Control* Panel (inside the [**Module**] control). Next to the [**Module**] control is the [**Channel**] control which specifies the current channel the Pixie Viewer is interacting with. The [**Module**] and the [**Channel**] are the target for all actions executed from the Pixie Viewer.

Proceed with the steps below to configure your system.

1. In the Calibrate tab, click on the Oscilloscope button.

   This opens a graph that shows the untriggered signal input. Click [**Refresh**] to update the display. The pulses should fall between about 1600 and 15000 on the left axis. If no pulses are visible or if they are cut off above 16384 or below 0, click [**Adjust Offsets**] to automatically set the DC offset. There is a control called [**Baseline**] on the Oscilloscope which can be used to adjust the DC offset for each channel. If the pulse amplitude is too large to fall in the display range, decrease the [**Gain**] in the Calibrate tab of the *Pixie4 Run Control* Panel. Since the offsets might drift, for example after changes in input count rate, it is useful to leave the display open and check the offsets once in a while.

2. In the Calibrate tab, input an estimated preamplifier RC decay time for Tau in µs then click on [**Auto Find**] to determine the actual Tau value for the current channel of the current module. Repeat this for other channels if necessary. The Tau finder works best for a Tau value from 20 µs to 200 µs.

3. In the Settings tab, click on [**Save**] to save the system parameters found so far. You can save the settings into either an existing settings file or a new file.

4. Click on the Run tab, set [**Run type**] to 0x301 MCA Mode, [**Polling time**] to 1 second, and [**Run time/timeout**] to 30 seconds or so, then click [**Start Run**]. After the run is complete, select the Analyze tab and click on the [**MCA Spectrum**] button. The MCA spectrum shows the MCA histograms for all four channels. You can deselect other channels while working on only one channel. You can do a Gauss fit on a peak by entering values in the [**Min**] and [**Max**] fields as the limits for a Gauss fit. You can also use the mouse to drag the Cursor A and B in the MCA spectrum to the limits of the fit. Enter the true energy value in the [**Peak**] field to calibrate the energy scale.

   If you are not getting a nice-looking spectrum, you may need to adjust some settings such as filter rise time and flat top etc. Refer to the User's Manual for details.

• Pixie-4 Control Panel

The *Pixie4 Run Control* Panel is displayed on the desktop after starting the IGOR experiment file "Pixie4.pxp". Using the four control tabs, arranged in logical order from left to right, you can set up the system, set run parameters, take data, and view the results.

Control Tabs:

Settings
Calibrate
Run
Analyze

It also has four buttons on the bottom of the panel:

[**Start System**] can be used to re-initialize the modules
[**More**] will show additional controls on the control panel
[**Help**] will open this help file, at the entry corresponding to the currently active tab

© XIA 2008. All rights reserved.

[**About**] will show software and hardware version information

- ## Settings

The *Settings* Tab of the *Pixie-4 Run Control* Panel contains parameters that control the operation of the Pixie-4. Most settings are changed on a channel-by-channel basis. Controls in the ModuleRegisterPanel affect the module as a whole, controls in the ChassisRegisterPanel are used to set up interactions between modules. The *Settings* Tab also contains controls used to load, save, copy, and extract settings.

If not all described controls are visible, click on the [**More**] button at the bottom of the panel.

Energy and Trigger Filter

In this group of controls you can set the rise times and flat top times for the [**Energy Filter**] and [**Trigger Filter**] of each channel. The units of time are µs. For a detailed description of filter parameters please refer to the User's Manual.
There are several filter [**Ranges**] for the energy filter; with a granularity of 0.027µs for filter range 1 (1-bit decimation), 0.053µs for filter range 2 (2-bit decimation), and so on. The trigger filter is always operated at the full ADC sampling rate. Its rise time can be varied between 27ns and 413ns. Its flat top however is valid between 0ns and 387ns. The trigger filter will most often use a flat top comparable with the average signal rise time. In applications with very short rise times a flat top of zero will give the best pileup rejection performance.

Employing a trapezoidal filter avoids the kind of ballistic deficit that occurs when a finite rise time signal is used in conjunction with a Gaussian shaper. The energy filter flat top time should thus be a little larger than the longest rise time expected. The output of the energy filter is sampled one decimated clock cycle before the end of the flat top, plus the signal arrival may jitter by up to one decimated clock cycle with respect to the decimated clock. You should therefore make the flat top two notches longer than the signal rise time.

The sum of energy filter rise time and flat top cannot exceed 127 decimated clock periods. If you type in a rise time or flat top value that violates this bound, the Pixie Viewer will adjust it accordingly.

The trigger threshold can be set in units of ADC steps. You can get an idea of what the noise in your system is by looking at the trace acquired in the Oscilloscope located on the Calibrate Tab. Use the mouse to zoom in on parts of the displayed trace(s) and estimate the noise.

Optimize Energy Filter

An optimization of energy filter settings is carried out by scanning all possible combinations of energy filter rise and flat top within their respective limits specified by the user, examining the energy resolution at each combination, and picking the optimal combination which gives the best energy resolution. Results of this optimization, including energy filter rise time, flat top and energy resolution, are stored in an output file whose name is specified by the user in the beginning of the run. Another file which has the same name as the output file but with a different extension (*.tmp) is used to store step-by-step intermediate results.

Click on [**Optimize**] to open the *Auto Optimization of Energy Filter* panel. Before starting the optimization, all conditions listed should be fulfilled first. This is to ensure that valid MCA spectra will be produced during the optimization and Gauss Fit of energy peaks on the MCA spectra will generate meaningful results. The four channels of a Pixie-4 module can be used to optimize four different Auto Scanning Limits at the same time when the same detector signal is split and input into the four channels. This would speed up the scanning of a large range of energy filter rise time and flat top.

© XIA 2008. All rights reserved.

You could change the Auto Scanning Limits of each channel by changing the [**Channel**] control on the left upper corner of the *Pixie4 Run Control* panel. At any time, the Pixie-4 module used to carry out this optimization is the current module, i.e. the module indicated by the [**Module**] control on the left upper corner of the *Pixie-4 Run Control* panel. So if you want to use a different module, change the [**Module**] number before you set the Auto Scanning Limits for each channel.

Pulse Shape Analysis

The [**Trace length**] and [**Delay**] values entered in this group of controls, both in units of µs, govern the waveform acquisition. [**Trace length**] is the total length of the acquired waveform, which can be set independent of the parameters in the Energy and Trigger Filter. You can use the delay parameter to move the trigger point within the trace: [**Delay**] measures the trigger time with respect to the beginning of the recorded trace. For ordinary data taking the trace lengths are up to 13.6µs for each channel. The waveforms will be read in 13.3ns increments from FIFO memory.
If event dead time is a concern, the trace length should be set as short as possible. In particular, if only energies and time stamps are of interest, the trace length can be set to zero.

The [**PSA Start**] and [**PSA End**] specify the trace range for Pulse Shape Analysis (PSA). Currently the Pixie-4 supports two types of PSA: XIA_PSA and USER_PSA. XIA_PSA reports the signal arrival time by measuring the time when the trace reaches a preset percentage level ("constant fraction") of its magnitude. The preset percentage threshold can be set defined in the ChannelRegisterPanel. The arrival time is relative to the starting time of the trace. So for XIA_PSA, the PSA Start and PSA End should be set to include the rising edge of the trace. USER_PSA is a user-defined PSA value, calculated by customized user code linked to the general software.

Action Buttons

[**Channel**] (Channel Register)	ChannelRegisterPanel
[**Module**] (Module Register)	ModuleRegisterPanel
[**Chassis**] (Chassis Register)	ChassisRegisterPanel
[**Copy**] (Copy settings)	CopyPanel
[**Extract**] (Extract settings)	ExtractPanel
[**Files/Path**] (Files and path)	AllFilesPanel
[**Load**] (Load settings)	
[**Save**] (Save settings)	

• ChannelRegisterPanel

= Settings -> Channel Register

Each channel has its own channel control/status register (CCSRA). Click a check box to set or clear particular bits. Setting the checkboxes will automatically modify the 16bit register shown in the Settings Tab. We give a brief description of all relevant bits here. In the *Channel Register Panel* they appear in top to bottom order.

Bit 0: [**Respond to group triggers only**]
This bit controls waveform acquisition. To stop the FIFO and store a waveform, a channel must receive a trigger. When bit 0 is cleared, only the triggers generated in this particular channel are used. When bit 0 is set, the triggers will be the distributed triggers from other channels or modules (also may include this channel).

Triggers are always distributed within a module: Any channel with "trigger enabled" will issue a trigger

© XIA 2008. All rights reserved.

for distribution, and any channel set to "group trigger" (bit 0 is set) will respond to the distributed trigger, including those issued by itself. A further control in the ChassisRegisterPanel determines how trigger signals are distributed between modules.

Bit 1: [**Measure individual live time**]
This bit will in almost all applications be opposite to bit 0. Its setting decides who asserts the live time control. When cleared, the DSP ensures that during the event interrupt no channel can generate another trigger and latch new event data, at least not after the coincidence time window (see HitEditPanel below). This setting is useful in master slave operation and almost in all cases where list-mode data are required. On the other hand, when channels are operating independently and only MCA information is needed (MCA mode), but not list mode data, then this bit should be set to achieve the highest throughput.

Note: This function is currently being debugged/revised.

Bit 2: [**Good channel**]
Only channels flagged as good will be read out. This setting has no bearing on the channel's capability to issue a trigger. There can be a triggering channel whose data are discarded. Channels not marked as good will not be included in the automatic offset adjustment.

Bit 3: [**Read always**]
Set this bit if you want a good channel to be read out even if it did not report a hit. If operated in group trigger mode you will always get a valid waveform. This way you can collect waveforms not biased by trigger requirements. If the signal on the channel crossed the trigger threshold (even if not trigger enabled) you also get a valid energy and timestamp. If bit 14 is set, you will get an estimate of the energy even when the signal does not cross the trigger threshold.

Bit 4: [**Enable trigger**]
You can switch on any channel's ability to contribute to the event trigger with this bit.

Bit 5: [**Trigger positive**]
For channels with triggering enabled, this causes triggering on the rising edge of the input signal when the bit is set, and triggering on the falling edge when the bit is not set. The core of the trigger/filter FPGA can only trigger on a rising edge. So, if the bit is not set the FPGA will invert the signal before storing it in the FIFO and sending it to the core.

Bit 6: [**GFLT required**]
In a larger experiment you may want to exercise control over which events to accept and which to reject
based on external logic. A logic signal inhibiting event triggers can be distributed on the PXI backplane. When bit 6 is set, a logic high on the GFLT (Veto) backplane line will inhibit latching data and issuing triggers for this channel.

Bit 7: [**Histogram energies**]
Switch on incrementing an energy histogram in the DSP's MCA memory with this bit. You can choose to have histogramming in list mode runs. The histograms will continue to be updated over multiple runs, started with the resume run command.

Bit 14: [**Estimate energy if not hit**]

© XIA 2008. All rights reserved.

If a channel is read out even if it did not report a hit (see [**Read always**]), its energy is normally reported as zero since there was no valid local trigger to capture the value of the energy filter. However, since the energy filter is computed continuously, setting bit 14 will capture the filter value based on the (last) group trigger distributed within the module or over the backplane. This might be useful for channels with occasional very small pulses (below the threshold), or possibly to capture energy estimates on piled up pulses.
Note that since the timing of the group trigger is not precise with respect to the non-triggering pulse, the energy reported is only a rough estimate. It might help to set the flat top time to a large value to make the capturing of the energy filter less time sensitive.

Bit 10: [**Compute constant fraction time**]
The DSP can use pulse shape analysis to compute a precise signal arrival time using the digital equivalent of a constant fraction discriminator. For this to work correctly the rising part of the signal should be fully contained in the recorded trace. The time computed is the arrival time after the start of the acquired waveform in units of 1/256th of an ADC sampling interval. This information can be used to replace the recorded channel time which is derived from a (digital) leading edge discriminator, cf the User's Manual.

The standard DSP code shipped with the Pixie-4 has some pulse shape analysis capabilities already built in. One of these functions is a digital constant fraction discriminator. It uses an input parameter, namely the [**Threshold**] percentage. The default value is 25% since it is a commonly used threshold fraction for this type of discriminator. The result of the computation is the time of signal arrival measured with respect to the start of the acquired waveform. The result is written into the channel header in the linear output data buffer, cf the User's Manual for details.

[**Integrator**]
This variable controls the event energy reconstruction:

0:	Normal code.
1:	Use energy filter gap sum only. This is equivalent to simply integrating over the pulse. It is useful for scintillator applications where event energies can be derived by setting the energy filter flat top long enough to cover the whole scintillation pulses.
2:	Ignore energy filter gap sum when reconstructing event energy. This is useful for step pulses whose amplitude is the difference between the high and low steps.
3-5:	Same as 1, but the energy is multiplied by a factor 2,4, or 8, respectively. This is useful for very fast pulses that do not have a lot of area under the peak and thus will show up only at the very low end of the spectrum.

- ## ModuleRegisterPanel

= Settings -> Module Register

The Module Register Edit Panel controls several module-wide parameters.

Edit Required Hit Patterns

In this section, you can select the Coincidence Pattern that defines which channels have to contribute to an event to make it acceptable. For example, you might be only interested in events in which only a single channel was hit, or in any event in which more than two channels contributed. Below is a description for functions within a module. Module coincidence functions are described under ChassisRegisterPanel.

For each event, the Pixie-4 first reads which channels contributed before reading any further data. This Hit Pattern is a 4 bit number, e.g. (0001) if only channel 0 contributed or (1111) if all channels

© XIA 2008. All rights reserved.

contributed. By setting the checkboxes you can require that an event must match one of the selected hit patterns to be accepted for further processing. Any or all hit patterns can be selected; if none is selected no events will be accepted, if all are selected any event will be selected. The hit pattern (0000) can be useful when sharing triggers between modules: traces etc will be recorded in the module for channels with the "Read Always" Bit set in the ChannelRegisterPanel even if there is no hit in the module itself. Setting the checkboxes will automatically modify the 16bit coincidence pattern shown in the Settings Tab, where it can also be edited directly (type "0x####" to enter a hex number ####).

An example shall illustrate this feature. Assume a single module connected to 4 detectors which observe a Na-22 source, emitting back to back 511keV gamma-rays from positron annihilation. Channels 0 and 1 are connected to one pair of back to back detectors and channels 2 and 3 are connected to a second pair of back to back detectors. You are interested only in gammas from positron annihilation. Thus a coincidence in channel 0 and 1 or a coincidence in channel 2 and 3 is required. If all 4 channels were in coincidence, that would be fine too. So, the acceptable hit patterns would be (0011), (1100) and (1111), where the right most digit indicates channel 0 and the left most is for channel 3. To achieve the desired behavior, you have to select the three acceptable hit patterns in the HitEditPanel by checking the appropriate boxes, and deselect all other hit patterns by not checking their boxes.

Note the difference between "hit" and "trigger": A "hit" occurs if a pulse goes over threshold and passes pileup inspection. "Hits" can not be disabled. A "trigger" means there is a "hit" plus a request for event processing is sent to the module's DSP unit. "Triggers" can be disabled on a channel by channel basis in the ChannelRegisterPanel. Channels will be recorded if they have a hit or their [**Read always**] bit is set in the ChannelRegisterPanel, but only "hit" channels will have valid energy and timing information.

Coincidence Window

Each channel with a pulse above threshold, whether trigger enabled of not, contributes to the hit pattern the moment the pulse is validated as not piled up (i.e. energy filter time after rising edge of pulse). However, if the channel is not trigger enabled, its contribution to the hit pattern is cleared after a fixed delay (~400ns). This is to ensure hits don't linger for channels that are not read out.

The hit pattern is read for comparison with the coincidence pattern about 50ns after the *first* pulse is validated. If several channels contribute to an event, the minimum coincidence window -- the time period in which (delayed) channels can contribute to the hit pattern -- is thus ~50ns.

A difference in peaking times between channels will cause even channels with simultaneous pulses to contribute to he hit pattern at different times. The software thus calculates the required additional window width to compensate for any such difference, displays it in the [**Required additional width**] field, and ensures that the [**User added width**] is at least this value. If longer delays between channels are expected from the physics of the experiment, this added width can be increased up to a value of ~200ms.

If the required additional with is later decreased by reducing the difference in peaking times, a smaller coincidence window is possible. However, to avoid modifying a large coincidence window intentionally set by the user, the [**User added width**] is never *decreased* automatically.

Notes:
1) Any added coincidence window width will increase the time required to process an event and thus reduce the maximum count rate.
2) In run types 0x100-0x203, pulses occurring *during the readout of data* (after the end of the coincidence window) are lost. The readout may take several microseconds, longer if waveforms are to be recorded.
3) The cut off at the end of the coincidence window is precise to within 13.3ns

© XIA 2008. All rights reserved.

Group Trigger Timestamp

Set this checkbox to record individual timestamps for all channels in group trigger mode, rather than the identical time stamps based on the (last) group trigger.

Normally, in acquisitions with shared group triggers, all channels record the (identical) timestamp of the (last) group trigger for this event. Since waveforms are captured based on the group trigger, this ensures that within a data record traces and timestamps are correlated. However, if no waveforms are recorded, there is no time-of-arrival information of possible delays between channels. Setting the checkbox preserves the time difference information. See the user manual for details.

Note that since only trigger enabled channels latch time stamps, if the box is checked, this number is only valid for trigger enabled channels.

Clover Addback

To support 4-fold clover detectors, there is a further option to sum energies from individual channels in events with more than one hit and bin them into an addback spectrum. This function is enabled with the checkbox [**Sum channel energies for addback MCA**]. The Pixie-4 module will thus generate 4 channel MCA and one addback MCA at the same time.
If clover addback is selected, there is the further option to bin into the individual channel MCAs either energies from all events, or only energies from events with hits in a single channel.

Clover addback is currently only supported in MCA mode.

- ## ChassisRegisterPanel

= Settings -> Chassis Register

The Chassis Setup Panel controls several system-wide parameters.

Backplane Options

Trigger share mode
As described for the ChannelRegisterPanel, if group trigger operation is requested, each channel will respond to a distributed trigger rather than its own local trigger. Triggers are always distributed within a module: Any channel that is enabled to trigger will contribute to the distributed trigger, and any channel in group trigger mode will (only) respond to the distributed trigger, including those issued by itself.
In addition, triggers can be distributed over the PXI backplane between modules. The field [**Trigger share mode**] controls this distribution: There are currently 4 modes:

In mode = 0, the module is disconnected from the backplane. Group triggers are only distributed between the 4 channels of the module.

In mode = 1, the module is connected to the backplane's wired-OR lines. This means that any module in mode = 1 issues triggers on common backplane lines, and every module in mode = 1 responds to these triggers.

Modes 2 and 3 are used in large chassis with more than 8 slots, where backplane lines do not support a wire-OR across PCI segment boundaries. A module in mode = 2 receives triggers from the right neighbor, builds an OR with its own triggers, and sends them to the left neighbor. The leftmost module thus receives a "daisy-chain OR" of triggers from all modules and should be set to mode = 3 to put these distributed triggers on shared backplane lines (PXI_TRIG0, PXI_TRIG1). All modules in mode 2 or 3 respond to triggers on these lines (i.e. at the same time). Note that the

© XIA 2008. All rights reserved.

backplane has to be configured to drive PXI_TRIG0 and PXI_TRIG1 from the leftmost segment to the other segments. In the XIA 18-slot chassis, this drive direction is hardwired. For National Instrument chassis, use the NI PXI explorer to configure the chassis backplane.

Notes:
Since other backplane lines are used for other Pixie-4 functions (e.g. PXI_TRIG3 for run synchronization), it is generally a good idea to configure the backplane to drive all PXI_TRIGx lines from the leftmost segment to the other segments. Also, for proper functioning of the run synchronization use mode 2 and 3 in systems with more than 7 Pixie-4 modules even if not sharing triggers.
Rev. B modules are always connected to the backplane (equiv. mode = 1).

Front Panel Input Settings
One option to connect an external GFLT (Veto) signal to the Pixie-4 system is to use the MMCX front panel connector on a Pixie-4 module. The GFLT signal is distributed via the PXI backplane to all modules, and - if enabled in the ChannelRegisterPanel - a channel only records events where the GFLT signal is logic 0 (0V). If the [**Use front panel for GFLT input**] checkbox is set, the current module will internally connect the front panel input labeled "DSP-OUT" to drive the GFLT backplane line. Only one device may issue the signal to the backplane. If the checkbox is set for one module, the function will automatically be disabled for all other modules. However, it is the user's responsibility to ensure that no other device in the PXI chassis issues signals on this line.
The GFLT signal must be a LVTTL signal, i.e. 0 = 0V, 1 = 3.3V.

A second option for the MMCX front panel input is to configure it to contribute to the STATUS line on the backplane. This is a wire-OR line connected to all modules in a backplane segment. If the MMCX input is high, the module will pull down the wire-OR line (this constitutes a logic 1 for this line). The state of the STATUS line can be recorded during event processing (see below)

PDM control
For making coincidence decisions between modules, it is recommended to use XIA's logic and preamp power module, the PXI-PDM. It is meant to reside in slot 2. The PXI-PDM has no PCI interface, and thus is programmed by a Pixie-4 module in the neighboring slot. To enable a module to program the PXI-PDM, the checkbox [**Module programs PDM to immediate left**] has to be set. The Pixie-4 module then sends the [**PDM control pattern**] to the PDM.

The PXI-PDM receives the Pixie-4's hit pattern through the PXI Star Trigger lines. A module can be set to not send out its hit pattern -- the PDM assumes a default (0000) -- by unchecking the box [**Send local hit pattern to PDM in slot 2**].
Note that due to the architecture of the PXI Star Trigger, a module in slot 2 must not send signals on this backplane pin. Therefore for a module in slot 2 this checkbox is automatically cleared.

The coincidence decision in the PDM is determined from the full 48bit hit pattern of the modules in slots 3-14 and the [**PDM control pattern**] as listed below. This decision list is implemented in the PDM's firmware with no claim to cover all cases. Please contact XIA to request additional cases or to obtain verilog source code to write custom PDM firmware.

Events are accepted if MC ==1. Logic operators are AND = &, XOR =^, OR = |, NOT = !. Operators in front of an array hit[a:b] or {a,b,c,d} are unitary reductions to a single bit (e.g. &{a,c,b} = a AND b AND c). Variables are defined as follows:

multi_all	= multiplicity of all channels (0 ... 48 in any particular event)
multi_req1	= multiplicity requirement 1. its value is 1 for PDMcontrol[0:3] = 1,4,7; 2 for 2,5,8; 3 for 3,6,9; 0 otherwise
multi_req2	= multiplicity requirement 2. its value is equal to PDMcontrol[0:3]
MxCy	= Hit bit of module x, channel y. Module 0 is in slot 3.
payy	= pair yy, counting from ch. 0 and 1 of Module 0. Channels are ANDed

© XIA 2008. All rights reserved.

```
    poyy        = pair yy, counting from ch. 0 and 1 of Module 0. Channels are ORed
    pxyy        = pair yy, counting from ch. 0 and 1 of Module 0. Channels are XORed
    hitNs       = an array of all channel N hit bits e.g. hit0s = (M0C0,M1C0,M2C0,...)
    Msum_all    = number of modules active in event
    Msum_g1     = number of modules in group (module 0 ... module 5) active in event

  case (PDM control pattern )
   // test and debug, overall multiplicity
   16'h0000: MC <= 0;      // always low
   16'h0001: MC <= 1;      // always high

   // overall multiplicity
   16'h001X:  MC <= (multi_all >= multi_req2);   // multi_req2 defined above, X = 1...F
   16'h002X:  MC <= (multi_all == multi_req2);   // X = 1...F


   // single module tests -- can actually be done in P4 with "local contributes to global" option
   16'h0100:  MC <= (M0C0 & !M0C1 & !M0C2 & !M0C3 );  // 1 specific channel hit, no others
   16'h0101:  MC <= (M0C0 &  M0C1 & !M0C2 & !M0C3 );  // 2 specific channels hit, no others
   16'h0102:  MC <= (M0C0 &  M0C1 &  M0C2 & !M0C3 );  // 3 specific channels hit, no others
   16'h0103:  MC <= (M0C0 &  M0C1 &  M0C2 &  M0C3 );  // 4 specific channels hit
   16'h0104:  MC <= (M0C0                          );  // 1 specific channel hit, others don't matter
   16'h0105:  MC <= (M0C0 &  M0C1                  );  // 2 specific channels hit, others don't matter
   16'h0106:  MC <= (M0C0 &  M0C1 &  M0C2          );  // 3 specific channels hit, others don't matter

   16'h0111:  MC <= ( (multi00 >= multi_req1) );          // 1 or more
   16'h0112:  MC <= ( (multi00 >= multi_req1) );          // 2 or more
   16'h0113:  MC <= ( (multi00 >= multi_req1) );          // 3 or more
   16'h0114:  MC <= ( (multi00 == multi_req1) );          // 1
   16'h0115:  MC <= ( (multi00 == multi_req1) );          // 2
   16'h0116:  MC <= ( (multi00 == multi_req1) );          // 3


   // 2 module tests
  16'h0200:  MC <= ( (M0C0 & !M0C1 & !M0C2 & !M0C3) | (M1C0 & !M1C1 & !M1C2 & !M1C3) );
     // 1 specific channel hit in either module, no others
  16'h0201:  MC <= ( (M0C0 &  M0C1 & !M0C2 & !M0C3) | (M1C0 &  M1C1 & !M1C2 & !M1C3) );
    // 2 specific channels hit in either module, no others
  16'h0202:  MC <= ( (M0C0 &  M0C1 &  M0C2 & !M0C3) | (M1C0 &  M1C1 &  M1C2 & !M1C3) );
      // 3 specific channels hit in either module, no others
  16'h0203:  MC <= ( (M0C0 &  M0C1 &  M0C2 &  M0C3) | (M1C0 &  M1C1 &  M1C2 &  M1C3)  );
      // 4 specific channels hit in either module, no others

  16'h0204:  MC <= ( (M0C0                          ) | (M1C0                          )  );
      // 1 specific channel hit in either module, others don't matter
  16'h0205:  MC <= ( (M0C0 &  M0C1                  ) | (M1C0 &  M1C1                  )  );
      // 2 specific channels hit in either module, others don't matter
  16'h0206:  MC <= ( (M0C0 &  M0C1 &  M0C2          ) | (M1C0 &  M1C1 &  M1C2          )  );
      // 3 specific channels hit in either module, others don't matter
  16'h0207:  MC <= ( (M0C0 & !M0C1 & !M0C2 & !M0C3) ^ (M1C0 & !M1C1 & !M1C2 & !M1C3)  );
      // 1 specific channel hit in either, not both modules, no others
  16'h0208:  MC <= ( (M0C0 &  M0C1 & !M0C2 & !M0C3) ^ (M1C0 &  M1C1 & !M1C2 & !M1C3)  );
      // 2 specific channels hit in either, not both modules, no others
  16'h0209:  MC <= ( (M0C0 &  M0C1 &  M0C2 & !M0C3) ^ (M1C0 &  M1C1 &  M1C2 & !M1C3)  );
      // 3 specific channels hit in either, not both modules, no others
  16'h020A:  MC <= ( (M0C0 &  M0C1 &  M0C2 &  M0C3) ^ (M1C0 &  M1C1 &  M1C2 &  M1C3)  );
```


© XIA 2008. All rights reserved.

```
      // 4 specific channels hit in either, not both modules, no others

   16'h020B:  MC <= ( (M0C0                                  ) ^ (M1C0                                  ) );
      // 1 specific channel hit in either, not both modules, others don't matter
   16'h020C:  MC <= ( (M0C0 &  M0C1                          ) ^ (M1C0 &  M1C1                          ) );
      // 2 specific channels hit in either, not both modules, others don't matter
    16'h020D:  MC <= ( (M0C0 &  M0C1 &  M0C2          ) ^ (M1C0 &  M1C1 &  M1C2          ) );
       // 3 specific channels hit in either, not both modules, others don't matter

    16'h0211:  MC <= ( (multi00 >= multi_req1 ) | (multi01 >= multi_req1) );   // 1 or more in either module
    16'h0212:  MC <= ( (multi00 >= multi_req1 ) | (multi01 >= multi_req1) );   // 2 or more in either module
    16'h0213:  MC <= ( (multi00 >= multi_req1 ) | (multi01 >= multi_req1) );   // 3 or more in either module
    16'h0214:  MC <= ( (multi00 == multi_req1 ) | (multi01 == multi_req1) );   // 1 each in either module
    16'h0215:  MC <= ( (multi00 == multi_req1 ) | (multi01 == multi_req1) );   // 2 each in either module
    16'h0216:  MC <= ( (multi00 == multi_req1 ) | (multi01 == multi_req1) );   // 3 each in either module

    16'h0221:  MC <= ( (multi00 >= multi_req1) ^ (multi01 >= multi_req1) ); // 1 or more in either, not both modules
    16'h0222:  MC <= ( (multi00 >= multi_req1) ^ (multi01 >= multi_req1) ); // 2 or more in either, not both modules
    16'h0223:  MC <= ( (multi00 >= multi_req1) ^ (multi01 >= multi_req1) ); // 3 or more in either, not both modules
    16'h0224:  MC <= ( (multi00 == multi_req1) ^ (multi01 == multi_req1) ); // 1 in either, not both modules
    16'h0225:  MC <= ( (multi00 == multi_req1) ^ (multi01 == multi_req1) ); // 2 in either, not both modules
    16'h0226:  MC <= ( (multi00 == multi_req1) ^ (multi01 == multi_req1) ); // 3 in either, not both modules

     16'h0230:  MC <= ( (pa00 | pa01) | (pa02 | pa03)  );  // one or more pairs out of 4 pairs ; a pair is two channels hit
     16'h0231:  MC <= ( (pa00 | pa01) ^ (pa02 | pa03)  );  // one or more pair in one module, not both
     16'h0232:  MC <= ( (pa00 ^ pa01) | (pa02 ^ pa03)  );  // exactly one pair in one or both modules
     16'h0233:  MC <= ( (pa00 ^ pa01) ^ (pa02 ^ pa03)  );  // exactly one pair in one, not both module

     16'h0234:  MC <= ( (po00 | po01) | (po02 | po03)  );  // one or more pairs out of 4 pairs ; a pair = at least one of two channels
   hit
     16'h0235:  MC <= ( (po00 | po01) ^ (po02 | po03)  );  // one or more pair in one module, not both
     16'h0236:  MC <= ( (po00 ^ po01) | (po02 ^ po03)  );  // exactly one pair in one or both modules
     16'h0237:  MC <= ( (po00 ^ po01) ^ (po02 ^ po03)  );  // exactly one pair in one, not both module


    // 3 module tests
    16'h0300:   MC <= ( (M0C0 & !M0C1 & !M0C2 & !M0C3) | (M1C0 & !M1C1 & !M1C2 & !M1C3) |
                       (M2C0 & !M2C1 & !M2C2 & !M2C3) );   // 1 specific channel hit in either module, no others
    16'h0301:   MC <= ( (M0C0 &  M0C1 & !M0C2 & !M0C3) | (M1C0 &  M1C1 & !M1C2 & !M1C3) |
                       (M2C0 &  M2C1 & !M2C2 & !M2C3) ); // 2 specific channels hit in either module, no others
    16'h0302:   MC <= ( (M0C0 &  M0C1 &  M0C2 & !M0C3) | (M1C0 &  M1C1 &  M1C2 & !M1C3) |
                       (M2C0 &  M2C1 &  M2C2 & !M2C3) ); // 3 specific channels hit in either module, no others
    16'h0303:   MC <= ( (M0C0 &  M0C1 &  M0C2 &  M0C3) | (M1C0 &  M1C1 &  M1C2 &  M1C3) |
                       (M2C0 &  M2C1 &  M2C2 &  M2C3) ); // 4 specific channels hit in either module, no others

    16'h0304:   MC <= ( (M0C0) | (M1C0) | (M2C0) );
          // 1 specific channel hit in either module, others don't matter
    16'h0305:   MC <= ( (M0C0 &  M0C1) | (M1C0 &  M1C1) | (M2C0 &  M2C1) );
         // 2 specific channels hit in either module, others don't matter
    16'h0306:   MC <= ( (M0C0 &  M0C1 &  M0C2) | (M1C0 &  M1C1 &  M1C2 ) |
         (M2C0 &  M2C1 &  M2C2) );   // 3 specific channels hit in either module, others don't matter

    16'h0311:    MC <= ( (multi00 >= multi_req1) | (multi01 >= multi_req1) | (multi02 >= multi_req1) ); // 1 or more in either
   module
    16'h0312:    MC <= ( (multi00 >= multi_req1) | (multi01 >= multi_req1) | (multi02 >= multi_req1) ); // 2 or more in either
   module
```


© XIA 2008. All rights reserved.

```
16'h0313:   MC <= ( (multi00 >= multi_req1) | (multi01 >= multi_req1) | (multi02 >= multi_req1) ); // 3 or more in either module
16'h0314:  MC <= ( (multi00 == multi_req1) | (multi01 == multi_req1) | (multi02 >= multi_req1) ); // 1 each in either module
16'h0315:  MC <= ( (multi00 == multi_req1) | (multi01 == multi_req1) | (multi02 >= multi_req1) ); // 2 each in either module
16'h0316:  MC <= ( (multi00 == multi_req1) | (multi01 == multi_req1) | (multi02 >= multi_req1) ); // 3 each in either module

16'h0330:  MC <= (   (pa00 | pa01) | (pa02 | pa03) | (pa04 | pa05)  ); // one or more out of 6 pairs ; a pair is two ch.hit
16'h0331:  MC <= ( ^{(pa00 | pa01) , (pa02 | pa03) , (pa04 | pa05)} ); // one or more pair in one module, not more
16'h0332:  MC <= (   (pa00 ^ pa01) | (pa02 ^ pa03) | (pa04 ^ pa05)  );    // exactly one pair in one or more modules
16'h0333:  MC <= ( ^{(pa00 ^ pa01) , (pa02 ^ pa03) , (pa04 ^ pa05)} ); // exactly one pair in one module

16'h0334:  MC <= (   (po00 | po01) | (po02 | po03) | (po04 | po05)  ); // one or more out of 6 pairs ; a pair is one of two ch. hit
16'h0335:  MC <= ( ^{(po00 | po01) , (po02 | po03) , (po04 | po05)} ); // one or more pair in one module, not more
16'h0336:  MC <= (   (po00 ^ po01) | (po02 ^ po03) | (po04 ^ po05)  );    // exactly one pair in one or more modules
16'h0337:  MC <= ( ^{(po00 ^ po01) , (po02 ^ po03) , (po04 ^ po05)} ); // exactly one pair in one module


// 4 modules
16'h0400: MC <= (&hit0s[3:0]);         //  1 specific channel hit in (each of) 4 specific modules, others don't matter
16'h0401: MC <= (&hit0s[3:0] & &hit1s[3:0]);  // 2 specific channels hit in (each of) 4 specific module, others don't matter

// 5 modules
16'h0500: MC <= (&hit0s[4:0]);           // 1 specific channel hit in (each of) 5 specific modules, others don't matter
16'h0501: MC <= (&hit0s[4:0] & &hit1s[4:0]);  // 2 specific channels hit in (each of) 5 specific module, others don't matter

// 6 modules
16'h0600: MC <= (&hit0s[5:0]);           // 1 specific channel hit in (each of) 6 specific modules, others don't matter
16'h0601: MC <= (&hit0s[5:0] & &hit1s[5:0]);  // 2 specific channels hit in (each of) 6 specific module, others don't matter

// 7 modules
16'h0700: MC <= (&hit0s[6:0]);           // 1 specific channel hit in (each of) 7 specific modules, others don't matter
16'h0701: MC <= (&hit0s[6:0] & &hit1s[6:0]);  // 2 specific channels hit in (each of) 7 specific module, others don't matter

// 8 modules
16'h0800: MC <= (&hit0s[7:0]);           // 1 specific channel hit in (each of) 8 specific modules, others don't matter
16'h0801: MC <= (&hit0s[7:0] & &hit1s[7:0]);  // 2 specific channels hit in (each of) 8 specific module, others don't matter

// 9 modules
16'h0900: MC <= (&hit0s[8:0]);           // 1 specific channel hit in (each of) 9 specific modules, others don't matter
16'h0901: MC <= (&hit0s[8:0] & &hit1s[8:0]);  // 2 specific channels hit in (each of) 9 specific module, others don't matter

// 10 modules
16'h0A00: MC <= (&hit0s[9:0]);         // 1 specific channel hit in (each of) 10 specific modules, others don't matter
16'h0A01: MC <= (&hit0s[9:0] & &hit1s[9:0]);  // 2 specific channels hit in (each of) 10 specific module, others don't matter

// 11 modules
16'h0B00: MC <= (&hit0s[10:0]);        // 1 specific channel hit in (each of) 11 specific modules, others don't matter
16'h0B01: MC <= (&hit0s[10:0] & &hit1s[10:0]);  // 2 specific channels hit in (each of) 11 specific module, others don't matter

//12 modules
16'h0C00: MC <= (&hit0s[11:0]);        // 1 specific channel hit in (each of) 12 specific modules, others don't matter
16'h0C01: MC <= (&hit0s[11:0] & &hit1s[11:0]);  // 2 specific channels hit in (each of) 12 specific module, others don't matter

// 13 modules
16'h0D00: MC <= (&hit0s[12:0]);        // 1 specific channel hit in (each of) 13 specific modules, others don't matter
16'h0D01: MC <= (&hit0s[12:0] & &hit1s[12:0]);  // 2 specific channels hit in (each of) 13 specific module, others don't matter
```


© XIA 2008. All rights reserved.

```
  // group tests
  16'h1020: MC <= (MSUM_g1 >= 2);      // 2 or more modules in first group, rest don't care
  16'h1021: MC <= ( (MSUM_g1 >= 2) | (MSUM_all >=3) );       // 2 or more modules in first group, OR at least 3 modules
overall

  // ARSA tests
  //M0: gamma pairs, M1, M2: beta pairs
 16'h1100:   MC <= ( (pa00 ^ pa01) & ( ^{po02,po03,po04,po05} )  );
        // only one pair (=2 hits) from M0 AND exactly one pair (>0 hits) from M1,M2
 16'h1101:   MC <= ( (pa00 | pa01) | ( |{po02,po03,po04,po05} )  );
       // at least one pair (=2 hits) from M0 OR at least one pair (>0 hits) from M1,M2
 16'h1102:   MC <= ( (pa00 ^ pa01) | ( |{po02,po03,po04,po05} )  );
       // only one pair (=2 hits) from M0 OR at least one pair (>0 hits) from M1,M2
  16'h1103:   MC <= ( (pa00 ^ pa01) );
       // only one pair (=2 hits) from M0, rest don't care
  16'h1104:   MC <= ( (pa00 | pa01) );
       //  at least one pair (=2 hits) from M0, rest don't care
  16'h1105:   MC <= ( ((pa00 & !po01) | (!po00 & pa01)) & ( ^{po02,po03,po04,po05} )  );
        // exactly one pair (2/0 or 0/2 hits) from M0 AND exactly one pair (>0 hits) from M1,M2
  16'h1106:   MC <= ( ((pa00 & !po01) | (!po00 & pa01)) | ( |{po02,po03,po04,po05} )  );
        // exactly one pair (2/0 or 0/2 hits) from M0 OR at least one pair (>0 hits) from M1,M2

  default: MC <= 1;
```

Module Coincidence Setup

Each module always tests the *Hit Pattern* of contributing channels against the user defined *Coincidence Pattern* (set in the ModuleRegisterPanel) before recording and processing an event. This is called the local coincidence test.
A global coincidence test can be applied using a PXI PDM in slot 2: Each module can send its hit pattern to the PXI PDM. The module in slot 2 defines which test the PXI PDM should apply (see above), and the PXI PDM outputs the result of the test on a shared backplane line.
In addition, each module can add the negative result of its local test to the global test, essentially vetoing those events that do not pass its local test for *all* modules. This allows one or more master trigger modules to influence acquisition for all modules. Limited module coincidence requirements can thus be applied even without a PXI PDM

With the checkboxes in the [**Module Coincidence Setup**], each module can thus be set to accept events if
a) only the local coincidence test is passed (check "local")
b) only the global coincidence test is passed (check "global")
c) either the local OR the global coincidence test is passed (check "global" and "local")
d) the global test AND all relevant local tests pass (check "global" and "local adds to global")

The [**Coincidence Pattern**] and [**Coincidence Window**] for each module are shown for reference only.

Examples:
1. To require a local coincidence of channels 0-1, 2-3, or both, set the coincidence pattern to 0x9008 in the ModuleRegisterPanel and check only the "local test" box in the Chassis Register Panel

2. To require coincidence of channels 0 and 1 in Module 0, and no other channel/module matters, in the ModuleRegisterPanel set the coincidence pattern in Module 0 to 0x8888 and in all other modules

© XIA 2008. All rights reserved.

to 0xFFFF. In the Chassis Register Panel, check the "global test" box for all modules and the "local adds to global" box for module 0. No PXI PDM is required.

3. To require at least 3 channels to be active in all modules, use a PDM module in slot 2 and set the [**PDM control pattern**] to 0x0013 for the module in slot 3. Make sure the [**Module writes control pattern ...**] box is checked for this module and the [**Send local hit pattern to PDM**] box is checked for all modules. Then check the "global test" box for all modules

- ## CopyPanel

= Settings -> Copy

This panel can be used to copy parameter settings from one module to another. The source module and channel are selected at the top of the panel. The parameters to be copied are organized into list box in the left-hand column. The right-hand column shows the destination channels and modules for the copy operation. The Items to copy shown on the Copy Panel and the actual variables to be copied are listed below.

Items	**Actual variables to be copied**
[**Gain**]	Gain [V/V]
[**Offset**]	Offset [V] and base percent
[**Filter**]	Energy Filter Rise Time and Flat Top, Baseline Cut
[**Trigger**]	Trigger Filter Rise Time and Flat Top, Trigger Threshold
[**FIFO**]	Trace Length, Delay, dT [µs], PSA Start, PSA End, CFD Threshold
[**ChanCSR**]	Channel CSRA and Channel CSRB
[**Coinc.**]	Coincidence Pattern, Coincidence Window
[**MCA**]	Cut-Off Energy, Binning Factor
[**TAU**]	Tau [µs]
[**Intergrator**]	Integrator
[**ModCSR**]	All module coincidence settings and backplane options except front panel GFLT

After selecting source, destination and parameters, click on the [**Copy**] button to execute the copy operation.

- ## ExtractPanel

= Settings -> Extract

This panel can be used to extract parameter settings from a file to selected modules and channels. The source file is specified at the top of the panel. Click on the [**Find**] button to locate the source file. Parameters to be extracted and destination modules or channels are selected in the same manner as in the CopyPanel. Click the [**Extract**] button to execute the operation.

- ## AllFilesPanel

= Settings -> Files/Paths

This panel gives you access to the underlying files of the Pixie-4 software. Usually, these files are already loaded in the memory of the Pixie-4 Viewer. You only have to change these files when you receive updates from XIA.

The directory locations are specified as complete (not relative) search paths: the DSP Path for the DSP code; and the FPGA Path for the trigger/filter FPGA configuration. Use a colon (:) as the separator between drive name, directory, and subdirectories. Do not use backslashes (\). For example

© XIA 2008. All rights reserved.

use "D:XIA:data" rather than "D:\XIA\data".

File and Path names should not exceed 80 characters.

- ## Calibrate

If not all described controls are visible, click on the [**More**] button at the bottom of the panel.

Analog Signal Conditioning

In the Analog Signal Conditioning section you can set the gain and DC offset for the selected channel and module. Note that the ADCs are dc-coupled to the Pixie-4 inputs, and thus compensation for any DC-offset is necessary. You will rarely have to set this manually, as the DC-offsets can be adjusted automatically through clicking on [**Adjust Offsets**] on the Oscilloscope.

The gain and offset settings are given in units of V/V and V, respectively. The voltage gain computed is the ratio between the pulse height at the module input to the pulse height at the ADC input. Note that the ADC has a 2.2V input range.

Histogram Control

This section shows the parameters controlling the operation of the multichannel analyzer built into the DSP memory. Energy values are calculated to 16-bit fixed-point numbers. This would correspond to a 64k spectrum.

To map the full energy range into the available 32k spectrum, one has to combine bins. At minimum, 2 bins have to be combined into one, so the [**Binning Factor**] has to be set to 1 (combining 2^1 bins). Higher binning factors can be useful for low count rate or low resolution applications.

If you want to see a certain range of the spectrum at higher resolution you can enter a [**Minimum Energy**]. This will discard all energies below the minimum and start binning the spectrum from the minimum energy (Bin 0 = Emin). The minimum energy is not applied in MCA runs.

Decay Time

The [**Decay Time**] is the exponential RC time constant of the preamplifier. It is required in order to properly calculate corrections to measured energy values. To set and measure the decay time, enter an estimated value then click on the [**Auto Find**] button. You can also enter a known good value directly in the control. The RC calibration needs to be performed only once for a given preamplifier. The result is then stored in the parameter database, and can be saved in the settings file by clicking on the [**Save**] button in the Settings tab.

Manual Fit

Manual Tau Fitting is done on a channel-by-channel basis. Clicking [**Manual Fit**] opens the *TauDisplay*. First, choose the channel on which you want to do the manual fit by changing the [**Channel**] control on the left upper corner of the *Pixie4 Run Control* panel. Then set proper [**dT**] on the *TauDisplay* panel, and click the [**Run**] button to read an untriggered trace for the chosen channel. The exponential fit range can be set either by putting the Cursors A and B on the trace or changing the [**Fitting_start**] and [**Fitting_end**] controls on the *TauDisplay* panel. Clicking the [**Do exponential fit**] button will perform the exponential fit on the portion of the trace specified by the fitting range. If possible, some of the baseline after the pulse should be included in the fit region. The fitted tau is going to be reported in the [**Fitted tau**] control and the deviation between the fitting curve and original ADC trace is shown in the [**Deviation**] control. Click [**Tau OK**] to download the Tau value to the Pixie-4

© XIA 2008. All rights reserved.

module.

Optimize Tau

Clicking [**Optimize**] opens a panel that allows automatic scanning of all Tau values within the scanning limit specified by the user, examining the energy resolution at each Tau, and picking the optimal Tau value which gives the best energy resolution. Results of this optimization, including the Tau value and energy resolution, are stored in an output file whose name is specified by the user in the beginning of the run. Another file which has the same name as the output file but with a different extension (*.tmp) is used to store step-by-step intermediate results.

Before starting the optimization, all conditions listed on the *Auto Optimization of Decay Time* panel should be fulfilled first. This is to ensure that valid MCA spectra will be produced during the optimization and Gauss Fit of energy peaks on the MCA spectra will generate meaningful results. The four channels of a Pixie-4 module can be used to optimize four different [**Decay Time Limits**] at the same time when the same detector signal is split and input into the four channels. This would speed up the scanning of a large range of Tau values. You could change the [**Decay Time Limits**] of each channel by changing the [**Channel**] control on the left upper corner of the Pixie-4 Run Control panel. At any time, the Pixie-4 module used to carry out this optimization is the current module, i.e. the module indicated by the [**Module**] control on the left upper corner of the Pixie-4 Run Control panel. So if you want to use a different module, change the [**Module**] number before you set the [**Decay Time Limits**] for each channel.

• Oscilloscope

= Calibrate -> Oscilloscope

The Oscilloscope shows 8192 untriggered ADC samples from the input for each channel. The time between samples can be set using the [**dT**] variable. The display is updated through its [**Refresh**] button. The DC offset of the preamplifier signal has to be compensated for in order to bring the DC-coupled input into the ADC range. The exact DC value has no bearing on the acquired spectrum and its origin, which is always at zero. The DC-adjustment is used only to ensure that the signals to be measured fall comfortably into the ADC range. When clicking the [**Adjust Offsets**] button, the Pixie-4 Viewer will set the DC offset to a percentage of the full ADC range specified in the [**Offset%**] control.

The offset calibration must be performed with the preamplifiers connected to the Pixie-4 inputs and with both the preamplifier power and detector HV switched on. One should also repeat the offset calibration each time measurement conditions change in any major way, e.g., when the count rate changes greatly. All such changes may influence the DC offset value of the preamplifier signal.

To analyze the noise spectrum of the acquired trace, click on the [**FFT Display**] button, which opens the FFTDisplay.

To view the effect of the digital filters applied to the ADC traces, click on the [**Show Filters**] button which opens the ADCFilterDisplay.

FFTdisplay

You can analyze the noise spectrum in the trace captured in the Oscilloscope, by observing the Fourier transform of the signal. For best results, remove any source from the detector and only regard traces without actual events. The chart shows a plot of amplitude vs. frequency. The plot is calibrated such that a sine wave with 100 ADC units amplitude (200 units peak-to-peak) will show up with an amplitude of 100. To convert a noise floor measurement into ADC units/sqrt(Hz) use the variable FFTbin displayed at the top of the chart, which tells the width of each frequency bin in the Fourier spectrum. The conversion from amplitudes to rms ADC units/sqrt(Hz) is accomplished by multiplying

© XIA 2008. All rights reserved.

with 1/sqrt(2*FFTbin). Now, observe that an ADC unit corresponds to 61µV. Using the known gain of the Pixie-4 you can convert the noise into an input noise voltage density measured in V/sqrt(Hz). Or, given a particular energy calibration, the noise density can be expressed as eV/sqrt(Hz).

If you click on the [**Apply Filter**] button, you can see the effect of the energy filter simulated on the noise spectrum.

ADCFilterDisplay

This graph shows the ADC trace for the selected channel and the response of the (slow) energy filter and the (fast) trigger filter together with an estimate of the trigger threshold. The display is updated through its [**Refresh**] button.

Notes:
1. For best representation of the filters, the filter lengths should be an integer multiple of the ADC sampling interval.
2. Only a simple trapezoidal difference filter is shown for the energy filter. In the actual pulse height calculation, corrections are applied that take into account the decay of previous pulses, the contribution during the flat top time, and long term baseline effects.

• Run

If not all described controls are visible, click on the [**More**] button at the bottom of the panel.

Run Control

This section sets run type, run time, and polling time

[**Run Type**]
This popup menu is used to set the run type to one of the following modes:

List Mode
List mode is the general data acquisition run. Waveforms, energies and time stamps are collected on an event-by-event basis. The data is stored in various formats (see section 3.6 of the user manual for details):

0x100	full event data (9 words), plus waveforms
0x101	full event data (9 words), no waveforms
0x102	compressed event data (4 words), no waveforms
0x103	compressed event data (2 words), no waveforms

Since available memory limits the number of events that each module can store in its buffer, the Pixie Viewer computes the maximum number of events. When the maximum is reached, the run is stopped and the buffer is read out. For a longer run in list-mode, you can request several spills, or buffer fills. For example, if you request a run with 10 spills, you will get 10 list mode buffers worth of data. At start of the first run all previous run history is cleared, e.g. MCA memory and run and live time information. The next nine sub-runs are started with a Resume Run command, which leaves previous run information intact. Run times and live times and spectra in MCA memory are incremented.

You can also manually adjust the maximum number of events stored before the run is stopped. Some data acquisition systems, which are geared towards event-by-event readout and are not able to handle large buffers, may benefit from the capability to reduce the maximum number of events per spill.

Fast List Mode
Fast list mode is an event-by-event data acquisition run without waveforms. Since no traces are read out, the data acquisition is faster than a regular list mode; however, no pulse shape analysis (PSA) values are available. There will also be no run statistics, no coincidence testing, and no checks if the

© XIA 2008. All rights reserved.

event buffer is filled faster than the event processing rate. So the average trigger rate must be kept well below the processing rate. Otherwise, the data from the remainder of the run will be corrupted. The data is stored in various formats (see section 3.6 of the user manual for details):

0x200	full event data (9 words), no waveforms
0x201	full event data (9 words), no waveforms
0x202	compressed event data (4 words), no waveforms
0x203	compressed event data (2 words), no waveforms

Fast List Mode is mainly a legacy run type, and only marginally faster (slightly shorter dead time from event processing) than a normal list mode run recording waveforms of zero length.

MCA Mode

MCA mode puts all modules into a typical spectrum-only acquisition mode in which there are no list-mode data required. The event data is not stored in the output buffer, but only used to calculate the energy for incrementing the spectrum. Runs end after the time specified in [**Run Time**] counts down to zero. The “Maximum no. of Events” control is set to zero for MCA runs since it is not used to end the run.
Any [**Minimum Energy**] entered in the Histogram Control is ignored for MCA runs.

[**Poll Time**]
The polling time indicates the time interval at which the Pixie-4 Viewer checks if the run in the selected modules has ended. If so, runs are stopped in all modules, if they have not stopped already, and the data are read out.

[**Run Time**]
This variable is used to indicate the total run time for MCA runs or the overall timeout limit for list mode runs. During a run, it counts down the remaining time. List mode runs end when the [**Number of Spills**] is reached or the [**Run Time**] is down to zero, whichever comes first.

List Mode Spill Settings

[**Number of Spills**]
The variable indicates the number of repeated runs. It is only used in list mode runs. Even if set to zero, there will be at least one spill. List mode runs end when the [**Number of Spills**] is reached or the [**Run Time**] is down to zero, whichever comes first.

[**Timeout**]
This variable sets the timeout for each individual spill. During a run, it counts down the remaining time. If it reaches zero, the current spill is ended by the host, and a new spill starts.

[**Events / Buffer**]
This variable indicates the maximum number of list mode events the Pixie-4 module can store in its buffer for each run. It is automatically set to zero (indicating unlimited events) for MCA runs.The number is calculated automatically if you change the run type, but it can be reduced manually.

[**Multiple Buffers per Spill**]
There are two options to use the external memory on the Pixie-4 to temporarily store several buffers on board and read them out in a fast block read. This method reduced readout dead time (because the block read is faster).

If the box [**32 buffers per spill**] is checked, the module stores 32 data buffers in external memory, which then are read out in a fast block read. Each readout counts as one "spill". The total number of events will be 32 times the [**Events / Buffer**] times the [**Number of Spills**].

If the box [**OR double buffers**] is checked, the module stores 16 data buffers in the first half of the

© XIA 2008. All rights reserved.

external memory, which then are read out in a fast block read while another 16 buffers are stored in the second half, and so on. Each readout of 16 buffers counts as one "spill". The total number of events will be 16 times the [**Events / Buffer**] times the [**Number of Spills**]. The modules continue to take data until the number of spills is reached and the host stops the acquisition.

If neither are checked, buffers are read one by one from internal memory. The total number of events will be the [**Events / Buffer**] times the [**Number of Spills**]. The readout deadtime will be higher in this mode.

External memory readout is not supported for Rev. B modules.

Synchronization

The first check box asks if all runs should [**Simultaneously start and stop**] in all modules. In almost all multi-module systems this will be the case and the box should be checked.
Synchronization signals are distributed over a PXI backplane line. If a Pixie-4 module is present in the crate, but not part of the current data acquisition setup, it might inhibit the synchronization setup. For setups with more than 7 modules in chassis with PCI bridges, make sure the [**Trigger share mode**] in the ChassisRegisterPanel is set to 2 and 3.

If you also want all timers in all modules to be reset to zero with the start of the next data acquisition run, click the box [**Synchronize clocks**]. For this feature to be useful all Pixie-4 modules should be operating from the same master clock as described in the user's manual.
Normally, once clocks are synchronized, they will stay synchronized until modules are power cycled. Therefore the checkbox is cleared at the end of the run, preserving time correlation between subsequent runs. In some cases it may be beneficial to resynchronize the clocks in every new run (i.e. reset the clock to zero for every new data file). To do so, check the [**in every run**] checkbox.

Output File

You can choose a [**Base name**] and a [**Run number**] in order to form an output file name. The run data will be written to files whose name is composed of both (i.e. base####.xxx). The run number is automatically incremented at the end of each run if you select [**Auto increment run number**] on the Data Record Options panel, but you can set it manually as well. Data are stored in files in either the MCA folder if the run is a MCA run or the PulseShape folder if the run is a List Mode run. These files have the same name as the output file name but different extension as described below.

".bin"
For list mode runs, buffer data are stored in a file with name extension ".bin". This is a binary file consisting of 16bit unsigned integers.

".dat"
For list mode runs, a summary of event data is stored in a file with name extension ".dat". This is an ASCII file. For long list mode runs with many spills, this file can become much larger than the binary file. If you do not need he ASCII data, you can disable the ".dat" file by unchecking the [**Auto process list mode data ...**] in the Data Record Options panel.

".mca"
For both list mode runs and MCA runs, MCA spectrum data are stored in a file with name extension ".mca" if you select [**Auto store spectrum ...**] on the Data Record Options panel. This is a binary file consisting of 32bit unsigned integers. It includes MCA data for all modules, ordered from channel 0 of module 0 to channel 3 of the last module

".set"
Module settings are stored in a file with name extension ".set" after each run if you select [**Auto store settings ...**] on the Data Record Options panel. Run statistics can also be extracted from this file. This

© XIA 2008. All rights reserved.

is a binary file consisting of 16bit unsigned integers. It is equivalent to the settings files saved from the Settings tab.

".ifm"
For both list mode runs and MCA runs, run statistics information are stored in a file with the extension ".ifm" if you select [**Auto store statistics ...**] in the Data Record Options panel.

Start Run

After setting all parameters, you can start a run to take data. During the run, the Run Time control shows the remaining time for MCA runs or overall time out count down for list mode runs. If you select multiple spills for list mode runs, the number of spills will also count down during the run.

For list mode runs, when the first module reaching the preset maximum number of events stops its run, it will also stop the runs in all other modules if Module Synchronization is enabled. Then the data buffer of each Pixie-4 module will be read out and saved into a file. If more than one spill is requested, the run will resume in all modules.

When the Run Time control counts down to 0, the Pixie-4 Viewer will issue a run stop command to stop the runs in all modules, followed by readout of data to file.

Stop Run

If you want to stop a run before it finishes by itself, you can click on this button to manually stop it. This will end runs in all modules and read out and save the data.

Data Record Options

This panel gives you several options for automating tasks after or during a run. They are all checked by default to ensure all data are saved for each data run.

[**Auto increment run number**] will increase the run number in the file name of the data files to avoid overwriting of files.
In addition, if this option is checked, Igor will test at the beginning of a run if the output file already exists (testing the .bin file for list mode runs and the .mca file for MCA runs). If the file exists, Igor will increment the run number and try again; after 20 tries it will add "_new" to the base file name and continue. This feature is intended to avoid overwriting previous files when Igor only remembers a run number from an earlier run, for example after a PC crash.

[**Auto store spectrum ...**] will store the spectrum data automatically after a run as a binary .mca file.

[**Auto store settings ...**] will store the run parameters automatically after a run in binary format (same as settings file). Run statistics can also be extracted from this file.

[**Auto process list mode data ...**] will extract the energies and timestamps from the binary data and save them in a ".dat" file in ACSII format. This may take a very long time for large data sets.

[**Auto store statistics ...**] will store the run statistics and run start/stop date and time automatically after a run in ASCII format in an .ifm file.

In addition, you can choose to automatically [**Update MCA every N seconds**] during MCA runs or every N spills during List mode runs. "N" can be selected in the variable control field. Run statistics will be updated at the same time. You can also manually update the MCA spectrum by clicking the [**Update**] button in the MCA Spectrum display.

© XIA 2008. All rights reserved.

Further, you can choose to automatically store data in [**New Files every N spills**] during List mode runs or every N seconds during MCA mode runs. "N" can be selected in the variable control field. Run statistics will be updated at the same time. This feature can be used to break up output data into several smaller files; since it will take some time to save the data, N should be set to a rather large value, i.e. at least several hundred spills or several thousand seconds.
Note that a *new* run will be started after saving, i.e. all run statistics will be reset, MCA spectra will be cleared, and clocks resynchronized if the [**in every run**] checkbox is set in the Synchronization section in the Run Tab. Thus enabling this feature is equivalent to manually starting several individual runs with N spills.

[**Do not parse list mode file ...**] will disable the automatic parsing of the list mode file after the run to build a list of pointers to individual events. This is required for Igor to display events in the List Mode Traces graph, but may take a long time for large files. If disabled, you have to manually load the file in the List Mode Traces graph before viewing traces and/or energies.

• Analyze

On the top left part of the Analyze tab shows the run time and the measured event rate for the selected module. The right part shows for each channel the live time and the input count rate. Note that the run time is the sum of time spent in sub-runs (called spills), but ignoring the time it took the host to read out the data from Pixie-4 modules. Similarly, the live time was measured only while one of the sub-runs was ongoing.

Besides the [**Update**] button to refresh the run statistics (only), there are four buttons in the middle of the tab opening the following graphs and panels:

MCA Spectrum

Pulse-height spectra accumulated in the internal Pixie-4 memory can be displayed after pressing the MCA Spectrum button. Pulse heights are computed to 16 bits precision, i.e. correspond to 64k spectra. As the memory allows for only 32k words per channel, bins have to be combined according to the Binning factor for each channel in the Histogram Control.

You can select the module you want to inspect and you can add or remove individual channel displays by clicking the MCA check boxes. The [**Sum MCA**] checkbox controls display of the clover addback spectrum. Gray fields indicate values can be edited.

The [**Fit**] menu allows you to make Gaussian fits to peaks in the histograms. The fit range can be set channel by channel in the [**Min**] and [**Max**] fields, or by placing cursors on the spectrum with the mouse. The [**Fit**] menu starts a fitting routing for one or all channels. The routine does take a constant background term into account, though its value is not displayed. The fit results that are displayed include the peak position, the number of counts in the peak, and its relative and absolute full width at half maximum (FWHM), calculated from the Gaussian fit. For best results be sure to extend the fit range to cover some of the constant background.

To calibrate your energy scale, you can after the fit type the true energy value into the field [**Peak**] and the scale will automatically be adjusted.

The [**Sum**] menu gives you three options of summing the spectrum. You can either sum the range defined by the [**Min**] and [**Max**] fields without any corrections, sum the range and subtract the background, or sum the entire MCA. Summing is performed on all four channels at the same time. The calculated Sum is displayed in the [**Peak Area**] field, and the MCA maximum of the sum range is displayed in the [**Peak**] field.

The [**Files**] popup menu allows to store individual spectra and read back stored spectra from disk.

© XIA 2008. All rights reserved.

There are three operations:
[**Save MCA to Igor text file**] will save the current MCA (4 channels) as a scaled wave with some added comments in a text file. This can be useful to import data into other applications
[**Read MCA to Igor text file**] will read back a Igor text file saved as above.
[**Extract MCA from binary file**] will read MCA data for the current module from a binary file, such as the .mca file automatically saved at the end of the run. The file is assumed to be in 32bit unsigned integer format.

You can [**Update**] the MCA during a run at any time. The [**MCA source**] field displays where the MCA was last read from, e.g. from the module's memory or a specific file.

The [**Zoom**] buttons above the graph can be used as shortcuts to zooming operations with the mouse. The [**Reset Scale]** button resets the scaling of the MCA to 1/bin, i.e. it undoes any calibration entered through the [**Peak**] field.

List Mode Traces

After a list mode run has finished, the acquired waveforms and event data can be displayed on an event-by-event basis in the List Mode Traces panel. The most recently acquired data file will be searched for the event requested in the [**Trace number**] field. The display will show the traces from the selected module, and the associated time stamps, energies and PSA values for those channels that reported a hit in this event. Also shown are the hit pattern and event timestamp. Traces and energies are scaled as 16-bit numbers. Note that the ADC traces shown in the Oscilloscope are raw 14-bit numbers, i.e. ADC traces have values divided by 4.

Other displays show the [**Event times**] and the [**Hitpattern**]. The last 4 bits of the hit pattern mark the channels recorded (bits 0-3 for channel 0-3) and bits 8-11 mark the channels reporting a hit (bits 8-11 for channel 0-3). Bit 5 shows the result of the local coincidence test, bit 6 the level of the "Status" line, and bit 7 the result of the global coincidence test. Bits 4, 12-15 are reserved.

In order to display traces from an earlier experimental run one needs to change the Data File name by entering it directly in the [**Data File**] control or clicking the [**Find**] button.

To see the response of the energy and trigger filters for the current event, click on [**Digital Filter**], which opens the Filter Display. This window is mainly intended for diagnostic purposes. For information how to use the Pixie-4 for more detailed pulse shape analysis, please contact XIA.

Filter Display

This graph is used to see the simulated response of the energy and trigger filters in an event. You can browse the leading edge trigger filter response and the energy filter response of individual events. The latter requires a trace length of at least twice the peaking time plus the gap time to be displayed. The trace is shown in red. The trigger filter is shown in blue, and the energy filter is shown in green.

Note that only a simple difference filter is displayed for the energy filter, without the corrections for baseline and decay of the pulse used in the energy calculations.

List Mode Spectrum

Pulse height spectra can be reconstructed from list mode data stored on the disk. The file shown in the [**Data File**] field will be processed and the resulting histograms will be displayed for the selected Pixie-4 module. Use [**Read**] after changing the data file to process the new data, and [**Histo**] to update the displayed spectrum. The full spectrum length is equal to 64k channels. Use [**No. of bins**] and [**Delta E**] settings to compress the spectrum such that it fits the display. Hint: use 8000 and 4 to see the full range of data, and then adjust these numbers to zoom into the range of interest. The number of bins

© XIA 2008. All rights reserved.

and the delta E variables are kept in memory for each channel individually. Be sure to select the channel of interest prior to changing these variables. Use the mouse to zoom in on peaks of interest.

The [**Fit**] menu allows you to make Gaussian fits to peaks in the histograms. The fit range can be set channel by channel in the [**Min**] and [**Max**] fields, or by placing cursors on the spectrum with the mouse. The [**Fit**] menu starts a fitting routing for one or all channels. The routine does take a constant background term into account, though its value is not displayed. The fit results that are displayed include the peak position, the number of counts in the peak, and its relative and absolute full width at half maximum (FWHM), calculated from the Gaussian fit. For best results be sure to extend the fit range to cover some of the constant background.

The [**Zoom**] buttons above the graph can be used as shortcuts to zooming operations with the mouse. The [**Reset Scale]** button resets the scaling of the MCA to 1/bin, i.e. it undoes any calibration entered through the [**Peak**] field.

All Run Statistics

The run statistics for all modules and channels are displayed in the *All Run Statistics* panel. It also shows the time and date of run start and stop and the source where the statistics were last read from. The information can be refreshed during a run by clicking the [**Update**] button, it can be saved to or read from an .ifm file with the [**Files**] menu.

The "DAQ Fraction" is an estimate of the fraction of the lab time a module is taking data. As described above, "Run Time" and "Live Time" measure only the time a module was actively taking data, but not the time to setup a run or the time a module is waiting for readout by the host. Thus "Run Time" is always smaller than the elapsed "Lab time" (i.e. the difference between date/time of run start and run stop). "DAQ Fraction" is defined as "Run Time" / (Current date/time - run start date/time) *100 and thus gives an estimate of the time lost for readout etc. (After a run is finished, the run stop time replaces the current time)

Note that "Run Time" is read from the module, while the date/time is obtained from the host computer to a precision of 1s. "DAQ Fraction" is therefore not a precise number.

In list mode runs, the time to read out one spill is about 0.025s per module. The DAQ Fraction will drop significantly if the time to fill the memory approaches this value. To maximize the DAQ Fraction, a) run in 32 buffer/spill mode, b) make sure the polling time is significantly smaller than the fill time, c) define only those channels actually used as "good", and d) minimize the amount of waveform captured or run in a compressed list mode to store more events in the same amount of memory.
For example, with 4 channels active in mode 0x103, the memory will hold 744*32 = 23,808 events, so at a count rate of 23kcps, the memory will fill in about 1s and be read out in less than 0.1s, which results in a DAQ Fraction of more than 91%

© XIA 2008. All rights reserved.